Marble Madness

Final Report

Yanda Huang, Jennifer Isaza, Divya Kulkarni,

Chris Song, Andrew Wong

May 9, 2018

1.0 Project Goals:

Figure 1.1 Foam bridges on white board.

The main objective of this project is to detect foam bridges (as shown above in Figure 1.1) and use Baxter to move these magnetic foam bridges to form a route from start to end. This route will be created such that when a marble is dropped a predetermined starting point it will traverse through the bridges and reach its final goal.

This is an important problem because it demonstrates that a robot is able to understand its environment and effectively place the blocks in a location that is suitable for a specific task. Our planning is based on a physics simulator, which provides an alternative to learning-based methods in order to accomplish this task. Although our project is in the context of a magnetic marble run game, on a higher level it can be applied to many different tasks and environments that robots might encounter.

We began by choosing a simulation framework to gain understanding of the planning required for this task. In this simulation environment, we are looking at different optimizers to use in our final system flow. Our main hardware components include the Baxter robot, a RealSense D435 camera, and a computer running ROS that communicates with both Baxter and the camera.

A key challenge is successfully moving Baxter's arms and the custom grippers to grip, remove, and place the blocks as planned. Another challenge for autonomous operation is writing the computer vision (CV) algorithms to detect the blocks on the whiteboard and transform the image coordinates into real world coordinates with respect to Baxter's position.

2.0 System Architecture:

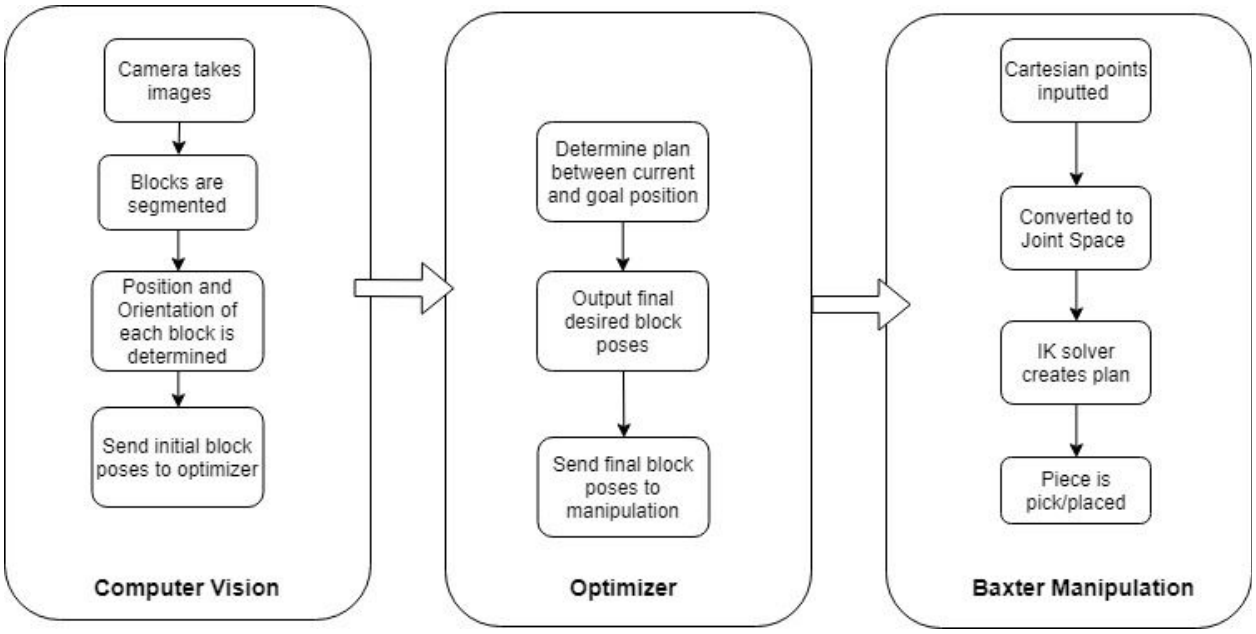

Figure 2.1 - System Architecture

As shown above in figure 2.1, there are three subsystems in our system architecture: vision, optimizer, and Baxter manipulation. Each subsystem's inputs, outputs and connections are described in the following section.

2.1 Vision Subsystem

Description: This subsystem processes an image of the whiteboard and returns information about the current configuration. The configuration will need to be adjusted for the marble to successfully fall from the start position to the end position. We are leveraging OpenCV libraries, but otherwise writing the code for this subsystem from scratch.

Inputs: RGB image of whiteboard with foam bridges.

Outputs: The bounds to be optimized. The width, height, 2d and 3d positions of each foam bridge. The location of the goal block.

Connections: Passes positions and classifications of foam bridges to the optimizer subsystem.

2.2 Optimizer Subsystem

Description: This subsystem finds the best configuration for the foam blocks, such that the ball will successfully arrive at the destination. This component is based on pre-existing project code, but heavily modified by us.
Inputs: The initial position and velocity of the ball, the radius of the ball, and bounds to be optimized. Additionally: width and height of each foam bridge, and the location of the destination.
Outputs: A valid configuration of foam bridges for the marble.
Connections: This subsystem sends a valid foam bridge configuration to the manipulation subsystem.

2.3 Baxter Manipulation

Description: Baxter picks a piece from the whiteboard and places it in the correct location. For this subsystem, we wrote code that utilized a IK solver routine provided Rethink Robotics.
Inputs: A valid foam bridge configuration (including x,y, and orientation of each piece) from the optimizer subsystem.
Outputs: Using Baxter's left arm to physically pick and place foam bridges.
Connections: None, this is the final subsystem in the pipeline.

3.0 Components Created

3.1 Optimizer

The optimizer uses an algorithm to find an optimal configuration for the foam blocks. An optimal configuration means the marble can travel down the track into the goal. The optimizer is integrated with Box2d, which is a physics simulator. Additionally, the optimizer minimizes a cost function, which is defined as the euclidean distance between the ball's final location got from the simulator and the destination. The optimization process runs for 500 iterations by default, and then send its results to the manipulation subsystem.

3.1.1 Algorithm/Method

The optimization algorithm we use is Differential Evolution. It is a derivative-free optimization method, spawning a family of random samples and mixing together

successful iterations. We used the default settings in the Scipy implementation, with a crossover probability of 0.7, a population size of 15, and a crossover strategy where the current best is mutated with the difference between two random samples in the population (this is a greedy variant of the Scheme DE1 proposed in [1]).

The communication interface with other programs is through a RESTful API with HTTP/1.1 protocol. The interface is built with Flask-RESTful. The visualization and simulation is done by modifying the Testbed component of Box2D. We injected more parameters into the built-in structure of Testbed so it can fully simulate our project.

3.1.2 Challenges

The main challenges we face is that the original code is written using the Metaprogramming technique, which means the code can modify itself at compile time. This technique is useful if the programmer has previous experience using metaprogramming. However, metaprogramming is difficult to understand for newcomers. It took a long time to understand the original code and then generalize it so it can handle all kinds of environments instead of only the hard-coded ones.

3.1.3 Other Methods

We also considered Open Dynamics Engine (ODE) as the simulator instead of Box2D. Additionally, we made a ROS package based on ODE. We picked Box2D finally because we did not find an optimizer integrated with ODE. The only solution we got from Chris Atkeson is integrated with Box2D, so we continued with that solution.

3.2 Computer Vision

The computer vision (CV) subsystem extracts the block poses from an image of the foam blocks on the board.

3.2.1 Segmentation

We are able to extract all the tracks by going through a very simple computer vision pipeline. Initially, a RGB snapshot will recorded by the Intel RealSense camera. Since the background will be mostly white, we can filter out the bulk of it with a simple thresholding operation in the RGB and HSV domain. To remove the shadow and other undesirable artifacts, we first remove any small blobs and morphologically “close” the image to make the images more concrete. Lastly, a media filter is used to remove any jagged contours in the image. A figure of the segmented image is shown below in figure 3.1.

Figure 3.1. Example of a segmented image.

3.2.2 Detection

Since all the marble tracks have their distinctive geometry, we can determine their general shape by their solidity, where solidity is simply the ratio of a blob's actual area to its convex hull. Therefore, for a shape like a rectangle, the solidity will be very close to one while the solidity for other irregular shapes will be lower. We have collected test data on all the tracks of different sizes and recorded their corresponding solidity threshold for accurate detection. With accurately extracted blobs, we can also compute their orientation with respect to the x-axis as well as their centroid location for grasping. figure of the post-processing is presented below in figure 3.2.

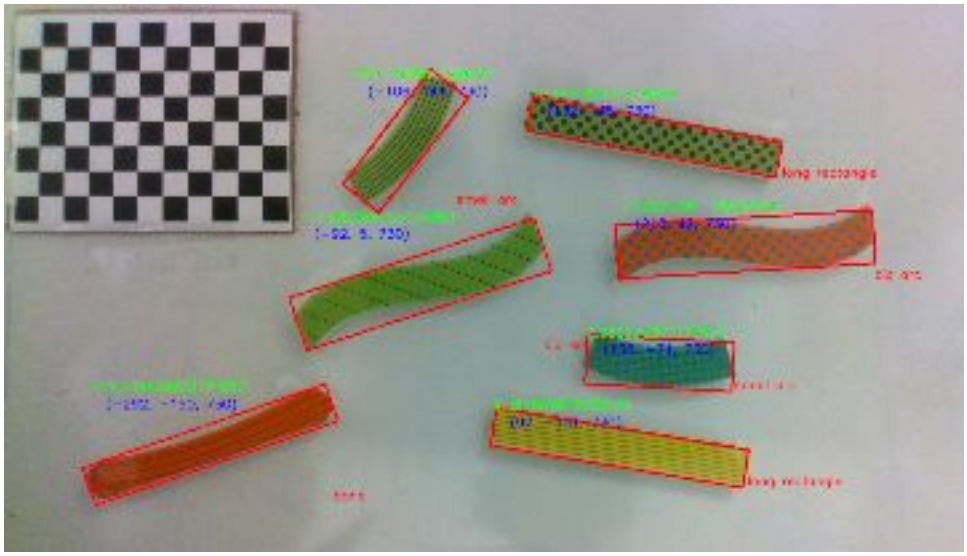

Figure 3.2. Post-processed image, showing block poses and classification.

3.2.3 Calibration

The camera must be calibrated for Baxter to know where the tracks are. We calibrated the RealSense with a checkerboard to extract both the intrinsic and the

extrinsic parameters to the camera. By doing so, we can relate image coordinates with world coordinates.

3.3 Mechanical Adjustments

Baxter has an electric gripper that must have attachment to grasp the desired objects. The standard attachments did not work for our specific objects because the foam bridges have a clear plastic “guard rail” that keeps the marble from falling off the piece. This would get crushed if used with the original grippers. We created a custom gripper that fits over the guard rail and contacts the foam piece for a stable grasp (see figure 3.4). There were some challenges in designing the part because the initial CAD models we found were not correct and had parts that were mirrored the wrong way. We then were able to contact Rethink Robotics to get access to the official CAD models that allowed us to design the grippers correctly. Additionally, we are creating a 3D printed mount for the RealSense Camera (see figure 3.5). We plan to position the camera at Baxter’s “chest” height so it can view a 3 ft x 2 ft area of the white board from about 3 ft away.

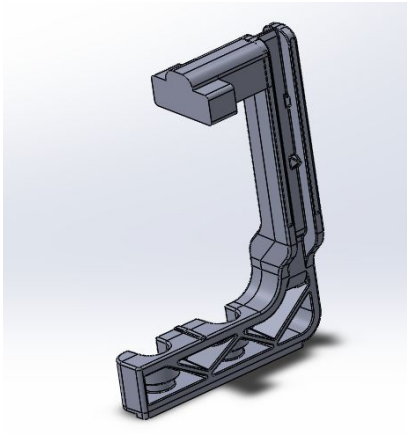

Figure 3.4. Custom Baxter gripper

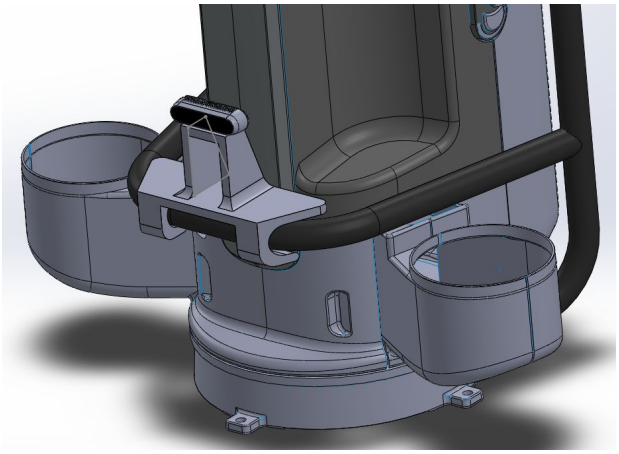

Figure 3.5. Camera Mount for Baxter

3.4 Baxter Manipulation

The manipulation subsystem uses coordinates from the CV and optimizer subsystems to plan a trajectory from the current grasper location to the foam block. We obtain the initial pose of the block from the CV, and the desired final block pose from the optimizer.

3.4.1 Planning method

The manipulation script requires a list of end effector cartesian coordinates and a wrist rotations represented as quaternions. We extract the initial and final coordinates

from the CV and optimizer outputs, respectively. The first coordinates are the initial block coordinates, followed by intermediate coordinates, and finally the desired block poses. The intermediate coordinates pull Baxter's grasper back, and prevent Baxter from failing to find an IK solution between the initi. For the initial block poses, we assume that the blocks are horizontal. Therefore, we provide a constant quaternion so that Baxter's graspers contact the top and bottom of the block. For the final block poses, the optimizer constraints its rotation output from $[-\frac{\pi}{2}, \frac{\pi}{2}]$, meaning that Baxter should never invert the block (it can rotate the block $\frac{\pi}{2}$ either way).

3.4.2 Challenges

MoveIt didn't work with Baxter, so we were forced to use the default IK solver provided by Rethink Robotics. We found that the IK solver often could not find an IK solution between end effector poses if they were too far apart. To remedy this issue, we decided to insert intermediate coordinates into the planning function. Additionally, we spent a significant amount of time on the transformation matrix and writing the planning code. The transformation matrix was difficult to get right since we had to manually measure point correspondences between the camera frame and Baxter's coordinates.

3.4.3 Other Methods

We tried using MoveIt! Interface to control Baxter. However, when using Moveit! interface, Baxter could form a joint trajectory plan, but could not actually execute the plan. We believe this is because Baxter failed to fetch the current joint state information. Since Baxter was not able to utilize MoveIt!, we switched to the default Baxter Interface for inverse kinematics planning.

4.0 Evaluation

To evaluate the CV subsystem, we compared the dimension and position of the foam bridges to the ground truth. In practice, we fed 5 images into the CV system, and compared the positions to measurements taken with a tape measure. We measured distances from the origin of the camera frame, which was at the center of our view. We repeated the process for three trials, for each type of block included in the marble kit.

We evaluated the optimization subsystem by running the optimizer on 5 different block configurations. Each block configuration was represented as an input image fed into the CV subsystem. For each image, we ran the optimizer 5 times. The optimization is considered successful if the ball arrives at the destination within the simulation.

To evaluate the entire system, we ran the entire pipeline for 5 iterations. Each time, we placed a marble at the top left track and let the marble travel down the track. If

the marble reaches the goal without falling off the track, the trial is considered successful. This tests the accuracy of the manipulation subsystem, as well as how well the entire system accomplished its goal.

4.2 Evaluation Results

Below are the evaluation results for the CV and optimizer subsystems, and the system as a whole.

4.2.1 CV Subsystem

Table 4.1. Error measurements for the CV subsystem

Trial	Block type	Error (cm)
1	Long rectangle	0.67
	Small arc	1.41
	Big arc	1.30
	Bone	1.17
2	Long rectangle	0.89
	Small arc	1.15
	Big arc	1.30
	Bone	1.17
3	Long rectangle	1.32
	Small arc	1.19
	Big arc	0.53
	Bone	0.82

4.2.2 Optimization Subsystem

The optimization subsystem was successful 25 times out of the 25 runs on 5 images.

4.2.3 Entire System

The marble made it to the goal successfully 3 out of the 5 trials.

5.0 Organization and Reflection

5.1 Division of Labor

We divided up the work on the project based on our previous skills and interests. Chris and Divya designed and 3D printed custom graspers and a camera mount. Chris worked on block detection using computer vision methods. Yanda modified the optimizer to output final block placements. Jen and Divya worked on Baxter's manipulation and planning, given coordinates from the CV and subsystems. Andrew worked with Chris on the CV subsystem and also assisted Divya and Jen with planning.

5.2 Challenges and Time Constraints

Specific portions of each subsystem took us longer to develop than we had anticipated. For the CV subsystem, we spent a significant amount of time tuning the thresholds for segmenting the block from the whiteboard. Additionally, it was challenging to find a mounting point for the camera that included a sizeable portion of the board. Lastly, inconsistent lighting conditions in the lab caused the CV subsystem to occasionally fail to detect the calibration checkerboard.

While we were integrating the manipulation and CV subsystems together, we were also writing the manipulation code. We couldn't move forward until we figured out the manipulation portion, causing a bottleneck for the project. Therefore, we could have had one or two team members develop the manipulation subsystem in parallel with the optimizer and CV subsystems, speeding up the integration process.

5.3 Lessons Learned

Several things can be changed if we were to do the project again. First, instead of doing the coordinate transformation manually, we can let ROS TF automatically calculate the transformation from camera to Baxter coordinates, simplifying our manipulation code. Second, the MoveIt interface is broken for the Baxter. If we could fix it, it would be much easier and faster to work with instead of operating on the raw Baxter interface. Third, the optimizer communicates with other components using a RESTful API, which is an outdated interface. A more future-proof solution would be using GRPC or GraphQL.

6.0 Video

Our video can be found here.

7.0 References

[1] Storn, R. and Price, K. (2018). *Differential Evolution - A simple and efficient adaptive scheme for global optimization over continuous spaces*. [online] Www1.icsi.berkeley.edu. Available at: http://www1.icsi.berkeley.edu/~storn/TR-95-012.pdf [Accessed 3 May 2018].